Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Engineering 38 (2012) 2272 – 2287

Procedia Engineering

www.elsevier.com/locate/procedia

ICMOC – 2012: International Conference on Modeling Optimisation and Computing

LOMPT: An efficient and Scalable Ontology Matching Algorithm

K. Saruladha[a], Dr. G. Aghila[b], B. Sathiya [a*]

[a]*Department of Computer Science, Pondicherry Engineering College, Puducherry, India*
[b]*Department of Computer Science, Pondicherry University, Puducherry, India*

Abstract

Ontology matching is an effective way to handle semantic heterogeneity among the ontologies. An ontology matching system with good efficiency and scalability is a challenge because of the monolithic nature and size of real world domain ontologies. In this paper, an efficient and scalable ontology matching algorithm called LOMPT (**L**arge **O**ntology **M**atching using **P**artitioning **T**echnique) is proposed. LOMPT consist of structure-based bottom up partitioning algorithm which decomposes the large ontology into a set of small partitions. Then the partition pairs across the ontologies are discovered based on the anchor distribution, where anchor is indentified by the proposed light weight string matcher SI-SUB. Finally the linguistic matcher V-DOC and structural matcher GMO process the partition pairs to find match results from the partition pairs. The example and experimental results depicts the efficiency and scalability of the proposed LOMPT system.

Keywords: Ontology matching; Ontology alignment; Ontology partition; Semantic heterogeneity

1. Introduction

* Corresponding author.
E-mail address: sathiya@pec.edu.

1877-7058 © 2012 Published by Elsevier Ltd.
doi:10.1016/j.proeng.2012.06.274

The semantic web is developed by the W3CSemantic Web Activity which aims at sharing and integration of data across various domains and enterprises. The vision of semantic web is made possible by the ontology which is the knowledge representations of the semantic web data. As the need and usage of the ontologies grow, the number of ontologies available also increases.

The semantic heterogeneity among these ontologies is more due to the decentralized nature of the semantic web where ontologies of same domain can be constructed by different experts. To resolve semantic heterogeneity and establish data integration, an effective way is the ontology matching. The goal of ontology matching is to discover semantically similar entity pairs among the input ontologies.

Although the field of ontology matching posses a good set of systems, still there are some open issues which is an obstacle for the good efficiency (time and space complexity) and effectiveness (match quality) of the matching system. In accordance with [1] one such open issue is matching large ontologies. Similarly according to the OAEI 2010 [2], the international **O**ntology **A**lignment **E**valuation **I**nitiative contest, only 50 % of the ontology matching systems are capable of matching large ontologies within one hour. This is because matching two ontologies requires each entity of one ontology to be compared against each entity of other ontology i.e. n^2 comparison, where n is the average size of the ontologies. The ontology with more than one thousand entities is considered as large ontology. When two large ontologies are matched, the number of comparison needed is very large, leading to more time and space complexity.

In accordance with [3], to handle large scale ontologies the scalability techniques used by the current system are as follows.

- Reduction of search space for matching
  - Early pruning of dissimilar element pairs
  - Partition-based matching
- Parallel matching
- Self-tuning match workflows
- Reuse of previous match results

Among the scalability techniques listed above this paper concentrates on partition-based matching technique .The partitioning technique divides the ontology into sub-ontologies called partitions. The ontology should be divided such that similar entities fall in same partitions and each partition of one ontology should be compared with few partition of other ontology. In comparison to the classical ontology matching technique, the number of entity comparisons are less in partition based ontology matching technique. This is because, in partition technique every entities of one ontology is compared with few entities of other ontology. This reduction in number of comparisons will lead to lesser time complexity and space complexity. Since advantage of partitioning technique is twofold we choose this technique to handle large ontology matching. The general design of the partition based ontology matching algorithm is shown in Fig. 1.

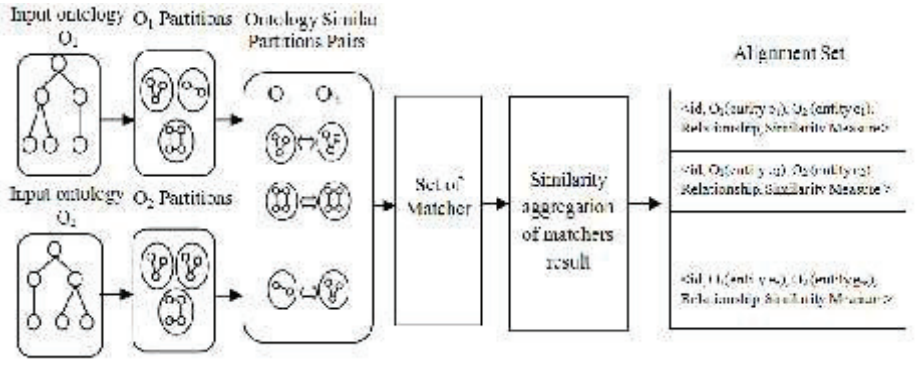

Fig. 1. General Framework of Partitioned Based Matcher System

The ontology matching systems which incorporated partitioning technique to handle large ontologies are Anchorflood [4], COMA++ [5], Falcon [6] and Taxomap [7]. Even though these systems can handle large ontologies they suffer from two drawbacks.

First, the partition technique used is either ontology language specific or too simple. Partition techniques of systems like Falcon and Taxomap are designed for RDF/OWL ontology languages, whereas sub graph of the ontology is considered as a partition in COMA++

Second, finding similar partition pairs is more time consuming (Falcon) or not accurate (COMA++, Taxomap and Anchorflood).

Hence, to overcome these drawbacks an efficient ontology matching algorithm LOMPT has been proposed. Specifically, a new effective and efficient structure based partition algorithm has been proposed which is neither language specific nor simple. Further a new efficient similarity measure to find similar partition pairs is also proposed.

The proposed ontology matching algorithm uses the general design of the partition based ontology matching algorithm as shown in Fig. 1. However the algorithm proposes new techniques to overcome the drawbacks of the existing systems. Specifically, the following proposals are made:

- A structural proximity calculation based on the neighbour of the entities which is used for partitioning of the ontologies is proposed. The time taken for computing structural proximity is less, since only entity pairs which shares common neighbours are considered for structural proximity computation. This reduction of computational time results in increased efficiency of the ontology matching process.
- A structure based partition algorithm to decompose large ontologies into set of partitions based on the above discussed structural proximity calculation is also proposed. The algorithm is linearly scalable i.e. O (n). Out of the four partitions based ontology matching systems mentioned above only Falcon and Taxomap uses a partition algorithm to decompose the ontology. The time complexity of the partition algorithm used in Falcon and Taxomap is O (n^2). Hence compared to the existing partition algorithms the proposed partition algorithm is more efficient.
- The proposed light weight string based matcher SI-SUB is used to find anchors (high similar entity pairs). These anchors are used to find similar partition pairs. Further entity level matching is restricted only to the similar partition pairs which reduce the search space and hence increase in efficiency.

The input ontology can be in OWL (**W**eb **O**ntology **L**anguage) format or RDFS (**R**esource **D**escription **F**ramework **S**chema) which can be converted to RDF graph [8]. The proposed large scale ontology matching algorithm LOMPT can be divided into three phases as follows.

Phase 1: The input ontology is divided into set of partitions based on the structural proximity between the entities which is calculated using neighbour of the entities. The partition algorithm divides the ontology using the structure proximity such that structurally similar entities are in same partitions.

Phase 2: After partitioning, the similar partition pairs are identified based on the anchor distribution across the two ontologies. The anchors are discovered by a light weight string matcher SI-SUB.

Phase 3: Only the similar partition pairs obtained above are further matched by any existing matchers to find the match results. Fig. 2 depicts the overview of the proposed ontology matching system LOMPT.

Section 2 discuss in detail about the existing large scale ontology matching system and its drawbacks. Section 3 discuss in detail about the proposed structural proximity calculation and partition algorithm. Section 4 presents the steps to find similar partition pairs. Section 5 explains the efficiency and effectiveness of the LOMPT using example. Section 6 depicts the experimental setup and results. Section 7 concludes the paper and list the possible future enhancement.

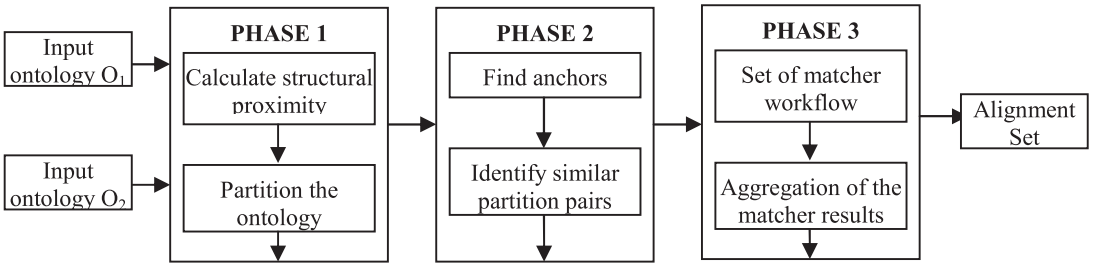

Fig. 2. Overview of the proposed ontology matching system LOMPT.

2. Related Work

As already mentioned, to handle large ontology matching the scalability techniques like early pruning of ontology, partitioning of ontology, parallel matching, self tuning and match reuse are adapted. QOM [9] and Peukert et al. [10] systems have used early pruning technique. Systems like Anchorflood, COMA++, Falcon and Taxomap incorporates the partitioning technique. In [11], Gross et al. used parallel matching technique to reduce the execution time. The self tuning technique to improve the matching is used in systems like RiMOM [12], Falcon and Peukert et al. The match reuse technique is utilized in COMA++ system.

Since this paper concentrates on partitioned based ontology matching algorithms only Anchorflood, COMA++, Falcon and Taxomap systems are discussed in detail.

Partitioned based matcher aims at partitioning the ontologies into set of partitions, such that the number of similar partition pairs across the ontologies is less. The system first decomposes the input ontologies into set of partitions. After partitioning, the similar partition pairs are identified between the two input ontologies. Then each similar partition pairs are processed by a set of matchers where the matchers can be arranged in sequential or parallel or in combined manner. After processing the similar partition pairs, aggregate the results of matcher's set and output the alignment set. In all partitioning based ontology matching system the large ontology is divided into set of smaller partitions. But the techniques used for partitioning differ in Anchorflood, COMA++, Falcon and Taxomap which is discussed in detail below.

The Anchorflood system adopts a dynamic partitioning technique which starts with a set of anchors (look alike entity pairs one from each ontology). One anchor is chosen at a time and the neighbour of the chosen anchor is collected. A partition (segment) pair is constructed from the above collected neighbours which is processed to produce match results. The algorithm is iterated until either all the collected neighbour entities is explored or no new match result is discovered.

The COMA++ incorporates the partitioned based matching by so called fragment based matcher and it works in two phases. The first phase, consist of 3 steps as follows:

- User choose fragment type viz UDF (User Defined Fragment), schema, subschema or shared fragment.
- Determination of fragments based on the selected fragment type.
- Identify the similar fragment pairs by light weight string matcher based on the fragment's root entity information.

In the second phase, the similar fragment pairs are further processed by a set of matchers to find the match results.

Unlike Anchorflood and COMA++, Falcon methodically partition the ontology based on the ROCK [13] clustering algorithm. The clustering algorithm ROCK uses depth information of the entities to

partition the ontology. After the partitioning, the similar partition pairs between the two ontologies are discovered based on the anchor (high similar match results) distribution across the partitions pairs. Anchors are found by processing the entire two ontologies using the light weight string matcher. Only the similar partition pairs are matched further by the existing set of matchers.

Taxomap ontology matching system has two algorithms as follows: Partition Anchor Partition (PAP) and Anchor Partition Partition (APP). PAP is used for matching structured vs. unstructured ontology whereas the APP is used for matching structure vs. structure ontology. In PAP first the structured ontology is partitioned based on the Falcon partition algorithm. Later the anchors (look alike entity pairs) are identified. Finally the unstructured ontology is partitioned based on anchor distribution such that anchors are tried to be localized on few partitions which will lead to less number of similar partition pairs. In APP, the anchors are identified and both the ontologies are partitioned such that the number of similar partition pairs is less.

2.1. Limitations of the related work

The drawbacks of the existing, partition based ontology matching systems are discussed below. The first drawback is identification of ontology partitions. Anchorflood ontology matching system performs dynamic partition of the ontology based on the input set of anchors (similar concept pairs) and the quality of the partition depends on the anchor input. In COMA++ a simple heuristic rule is used to partition the ontology which often leads to too many or too few partitions. The partition algorithm used in Falcon and Taxomap are more time consuming and specific to RDF ontology language. The second drawback is the structural proximity calculation which is used for partitioning of the ontology. COMA++ doesn't consider the structural proximity between the concepts for partitioning. Falcon and Taxomap make use of only the concepts depth information.

3. Proposed Ontology Partitioning

In this section, first the measure of structural proximities is introduced. Then the partition algorithm which is used to decompose the ontology into set of partitions is presented.

3.1. Structural proximity

The structural proximities between the entities are based on the shared neighbour of the entities. The neighbours of entity are the descendants, ascendants and siblings of the entity, i.e. the entity's children, entity's parents , entity's siblings and entity itself constitutes the neighbours of the entity. The neighbour of the entity denoted by N(e) is defined as follows.

$$N(e) = \{w \in E \mid (w \in \text{descendant}(e)) \text{ or } (w \in \text{ancestor}(e)) \text{ or } (w \in \text{sibling}(e))\} \text{ and } \{e\} \quad (1)$$

The structural proximity defined below is derived from the Tversky psychological model [14]. Let e_1, $e_2 \in E$ (Entity set), the structural proximity denoted by $\delta\alpha$ (e_i, e_j) is defined as

$$\delta\alpha(e_i, e_j) = \frac{|N(e_i) \cap N(e_j)|}{|N(e_i) \cap N(e_j)| + \alpha |N(e_i)| + (1-\alpha)|N(e_j)|} \quad (2)$$

where N(e) is defined based on equation 1. When an entity of a partition shares a similar structure with one of its neighbours, their computed intra ontology similarity will be large. The parameter α in equation 2 is calculated as follows.

$$\alpha(e_i, e_j) = \frac{N(e_i)}{|N(e_i) + N(e_j)|} \quad \text{when } |N(e_i)| \leq |N(e_j)| \quad (3)$$

$$\alpha(ei,ej) = 1 - \frac{N(ei)}{|N(ei)+N(ej)|} \quad when\ |N(e_i)| > |N(e_j)| \tag{4}$$

Another structural proximity computation is also derived from the Tversky psychological model under the assumption that each entity has equal importance in the ontology. Let e_i, $e_j \in E$ (Entity set), the structural proximity denoted by δ (e_i, e_j) is defined as

$$\delta(ei,ej) = \frac{N(ei) \cap N(ej)}{(N(ei)+N(ej))/2} \tag{5}$$

In total, (n-1) entity pairs are possible for ontology with n entities. In the proposed system only the entity pairs which have common neighbours are considered for computing structural proximity and hence the execution time is reduced.

3.2. Partitioning algorithm

The partition algorithm decomposes the ontology into set of partitions based on the structural proximity defined above. The ontology partitioning helps to reduce the search space by eliminating the dissimilar partition pairs across the ontologies from further match processing. The proposed partition algorithm is an agglomerative (bottom up) hierarchical algorithm extended from the partition algorithm AHSCAN [15] (Agglomerative Hierarchical Structure Clustering Algorithm for Networks). The AHSCAN is a linearly scalable clustering algorithm in the area of networks. Initially each n entity in the ontology is formed into n partitions. Then the partitions are merged in the bottom up fashion based on the inter ontology partition similarity which is defined below.

$$sim(p1,p2) = \frac{\sum_{i=1}^{k1}\sum_{j=1}^{k2}\delta(e1i,e2j)}{|k1||k2|} \tag{6}$$

where p_1 and p_2 are partitions, k_1 is the number of entities in p_1, k_2 is the number of entities in p_2 and δ (e_{1i}, e_{2j}) is the structural proximity between entity $e_{1i} \in E_1$ and $e_{2j} \in E_2$.

The intra cohesion of a partition is the structural closeness between the entities of the partition. The average intra cohesion value which is used to measure quality of partition set is denoted by Q(ps) and defined as follows.

$$Q(ps) = \frac{\sum_{i=1}^{m} Q(pi)}{n} \tag{7}$$

where ps is the partition set for which the quality is calculated, p_i is the i^{th} partition is ps, m is the number of partitions in ps, n is the number of entity in the ontology and $Q(p_i)$ is determined by calculating the structural proximity between all possible entity pairs in the partition p_i.

Fig. 3 depicts the proposed partition algorithm. As shown in figure the inputs to the algorithm are the ontology and the hash table containing the structural proximity value of the entity pairs. The proposed partition algorithm takes place in two steps. In the first step, the partition algorithm construct n partitions where each partition consists of only one entity and n is the number of entities in the ontology. In second step, the partition algorithm iteratively merges the structurally similar partition pairs of the partition set to form temporary partition set. The algorithm terminates if the average cohesion value of the temporary partition set is less then threshold β(to be experimentally determined) . Otherwise the algorithm store the temporary partition set as the partition set and continue to merge the partitions further. The structural proximity value in the hash table is removed if the entity pair is merged into same partition. Since the

structural proximity decreases as the partition size increases, the CutOff value should be reduced as the partition size increases. The proposed partition algorithm terminates when the number of partition in the partition set reaches one or no more structural proximity values in the hash table or the average cohesion value is less than the threshold β.

```
PartitionSolution PartitionAlgorithm (Ontology graph G, HashTable H)
//Partition initialization
For each entity e_i in G
        Create partition p_i with entity e_i
        Add p_i to PartitionSet
End For
While (Size (PartitionSet) > 1)
        //Partition merging
        For each partition pairs of the PartitionSet
                If sim(p_i,p_j) > CutOff
                        Merge the partition pair to form a new partition
                        Add the new partition to the TempPartitionSet
                Else
                        Add the partition pair to the TempPartitionSet
                End If
        //Termination Condition
        If (TempPartitionSet != PartitionSet )
                If Q(TempPartitionSet) > β
                        PartitionSet = TempPartitionSet
                Else
                        Break      //Terminate the algorithm
                End If
        End If
        //Returns true if there is no entry in the hash table H
        If  NoMoreMerge() then
                break
        End If
        Update Cutoff
End While
Return PartitionSet
```

Fig. 3. Proposed Partition Algorithm

In comparison the existing AHSCAN clustering algorithm differs from the proposed partition algorithm in four ways. First, in the existing algorithm an entity (v) is the neighbour of other entity (w) iff their exist edge between them. In the proposed algorithm an entity (v) is the neighbour of other entity (w), iff entity (v) is the descendants or ancestors of the entity w. Two closely related sibling entities don't have an edge between them and hence not considered as neighbour in the existing algorithm which should not be done. Second, we have used our proposed method for calculating the structural proximity. Theoretically it is evident that the proposed structural proximity method is efficient then the existing method. Third, the existing algorithm uses the parameter modularity, a quality measure for network clustering to find the best partition solution. In the proposed algorithm we have defined quality measure Q(ps) to determine the partition solution's quality. Fourth, the existing algorithm stores the hierarchy of

the partition set in a bottom up fashion and best partition pair is chosen based on the modularity value. This will lead to insufficient memory for large ontology. In the proposed algorithm the hierarchy of partition set is not stored. Only one partition solution is stored and upon merging if the average cohesion value goes beyond β, the merging is undone and the algorithm terminates.

4. Identifying Similar Partition Pairs

This section discusses the steps required to identify the similar partition pairs across the two ontologies. The proposed light weight string matcher SI-SUB is used to find anchors distribution across the two full ontologies. Then the similar partition pairs are identified based on the anchor distributions between the partitions.

4.1. Anchors identification

The alignments with high similarities are defined as anchors. For the proposed system a simplified version of I-SUB [16] called SI-SUB is used to find the anchors. Let p and q be description of two entities, the similarity between p and q is defined as follows.

$$SI\text{-}SUB(p,q) = comm(p,q) - diff(p,q) + wrinkler(p,q) \quad (8)$$

where comm(p,q) stands for the number of common characters between p and q scaled with the string length, diff(p,q) for the number of uncommon charactes between p and q scaled with the string length and winkler(p,q) [17] for the improvement of the result by using a correction coefficient. If the similarity between two descriptions is greater than a pre-defined value μ then SI-SUB would consider the two entities containing those two descriptions as a candidate anchor.

4.2. Finding similar partition pairs

The similar partition pairs are found based on the anchors distribution. A partition pair is said to be more similar if they share more anchors. Let PS_1, PS_2 be two sets of partitions from two ontologies O_1, O_2. The function anchor (p_{1i}, p_{2j}) returns the number of the anchors shared between two partitions p_{1i} and p_{2j} where $p_{1i} \in PS_1$ and $p_{2j} \in PS_2$. The similarity denoted by pm_sim(p_{1i}, p_{2j}) between p_{1i} and p_{2j} is defined as follows.

$$pm\_sim(p,p') = \frac{2.anchor(p,p',Q)}{\sum_{pk \in P} anchor(pk,p',Q) + \sum_{p'k \in P'} anchor(p,pk',Q)} \quad (9)$$

The partition pair whose similarity value is greater than the pre-defined value η is called as similar partition pair. The entity level matching is restricted only to similar partition pairs which will reduce the search space and thus decrease in execution time.

After finding the similar partition pairs each similar partition pairs can be processed by a set of available matcher to find the match result. The matcher can be based on linguistic or structural or instance features of the ontology. The matcher workflow can be sequential or parallel or in mixed fashion. The proposed ontology matching algorithm LOMPT discussed so far is outlined in Fig.4.

AlignmentSet Algorithm **LOMPT** (O_1, O_2)
//O_1 and O_2 are the two large input ontologies to be matched
Step 1: Preprocess the ontologies O_1 and O_2.
Step 2: Calculate the structural proximity between the neighbours of the entities in O_1 and O_2.
Step 3: Partition the ontologies O_1 and O_2 into set of partitions.
Step 4: Using light weight string matcher SI-SUB, the anchors set of O_1 and O_2 are identified.
Step 5: Identify the similar partition pairs across the two ontologies based on the anchors distribution.
Step 6: Each similar partition pairs are processed by the existing set of matchers to find the match results.
Step 7: Aggregate the match results from the set of matchers to output the alignments.

Fig. 4. Proposed large scale ontology matching algorithm LOMPT

5. Comparative Example

The existing partition algorithm of the FALCON and proposed partition algorithm is compared with the sample ontology O (Fig. 5(a)) and O` (Fig. 5(b)). Fig. 6 depicts the anchor distribution between the ontology O and O` taken from [7].

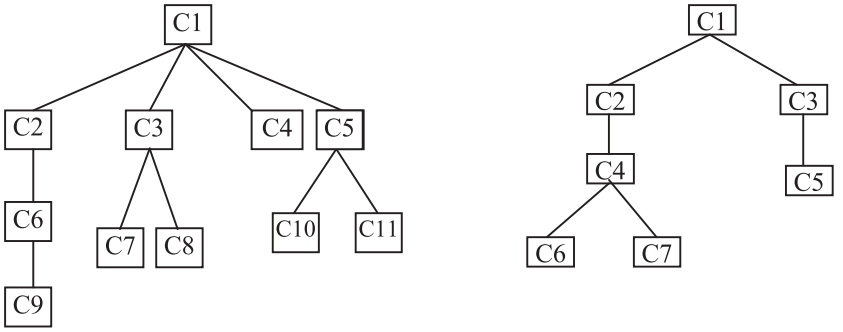

Fig. 5. (a) Sample ontology O; (b) Sample ontology O`

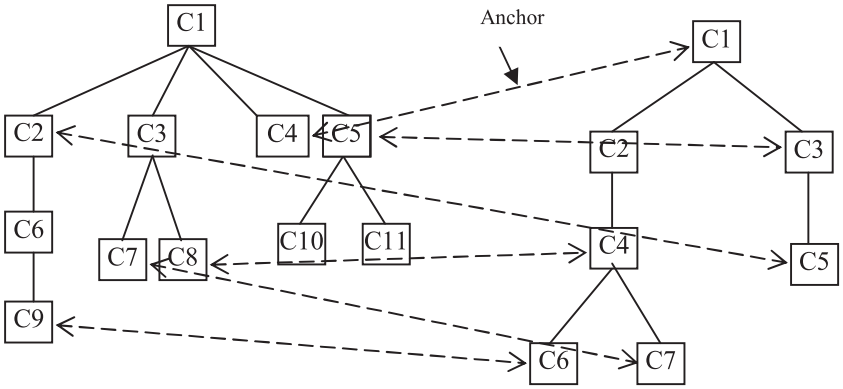

Fig. 6. Anchor Identification

5.1. *Existing Partition Algorithm*

Let c_i, c_j be two classes in a given ontology O, the structural proximity of the FALCON between c_i and c_j is defined as:

$$prox(ci,cj) = \frac{2.depth(cij)}{depth(ci) + depth(cj)} \quad (10)$$

where c_{ij} is the common superclass of c_i and c_j and depth(c_i) gets the depth of c_i in the original class hierarchy. Using equation 10, the structural proximity values for ontology O and O` are calculated.

The FALCON partition algorithm divides the ontology O into three partitions. Partition p1 contains {C1, C2, C4} concepts. Partition p2 contains {C6, C9} concepts. Partition p3 contains {C3, C5, C7, C8, C10, C11} concepts. The ontology O` is divided into two partitions. Partition p`1 contains {C1, C2, C4, C6, C7} concepts. Partition p`2 contains {C3, C5} concepts. Fig. 7(a) and Fig. 7(b) depicts the set of partition output of FALCON partition algorithm on ontology O and O`.

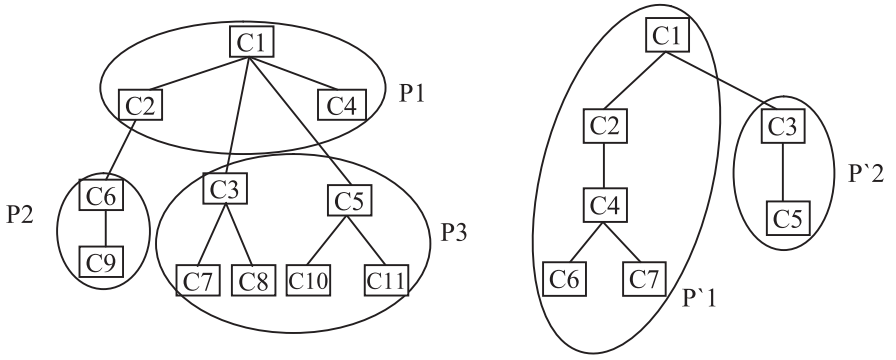

Fig. 7 (a) Existing system partition of ontology O; (b) Existing system partition of ontology O`

The possible pairs of ontology partitions are (p1, p`1), (p1,p`2), (p2,p`1), (p2,p`2), (p3,p`1) and (p3,p`2). Based on the anchor distribution the similar partition pairs are found using equation 9 and the values obtained are (p1,p`1) = 0.33, (p1,p`2) = 0.5, (p2,p`1) = 0.4, (p2,p`2) = 0, (p3,p`1) = 0.57 and (p3,p`2) = 0.4 . Let us assume cutoff η to be 0.1, so the pairs (p2,p`2) will be eliminated from further process of matching ie., only pairs (p1,p`1), (p1,p`2), (p2,p`1),(p3,p`1) and (p3,p`2) will be given as input to V-DOC.

5.2. Proposed Partition Algorithm

Using equation 5 the LOMPT structural proximity values for ontology O and O` are calculated. The LOMPT partition algorithm divides the ontology O into three partitions as shown in Fig. 8(a). Partition p1 contains {C1, C4, C5, C10, C11} concepts. Partition p2 contains {C2, C6, C9} concepts. Partition p3 contains {C3, C7, C8} concepts. The ontology O` is divided into two partitions as shown in Fig. 8(b). Partition p`1 contains {C1, C2, C3, C5} concepts. Partition p`2 contains {C4, C6, C7} concepts.

The possible pairs of ontology partitions are (p1,p`1), (p1,p`2), (p2,p`1), (p2,p`2), (p3,p`1) and (p3,p`2). Based on the anchor distribution the similar partition pairs are found using equation 9 and the

values obtained are (p1,p`1) = 0.66, (p1,p`2) = 0, (p2,p`1) = 0.4, (p2,p`2) = 0.4, (p3,p`1) = 0 and (p3,p`2) = 0.5 . Let us assume k =2 and cutoff η to be 0.1, so the pairs (p1,p`2) and (p3,p`1) will be eliminated from further process of matching i.e., only pairs (p1,p`1), (p2,p`1), (p2,p`2) and (p3,p`2) will be given as input to V-DOC.

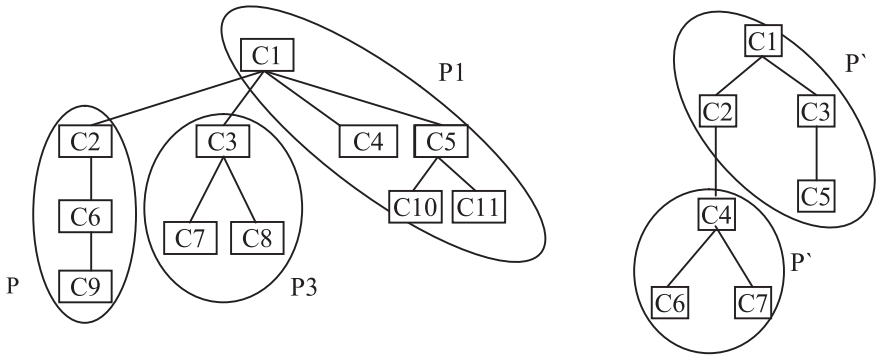

Fig. 8 (a) Proposed system partition of ontology O; (b) Proposed system partition of ontology O`

5.3. *Comparison of Existing and Proposed Partition Algorithm*

The partition algorithm of the proposed system is better than the existing system for two reasons. First, the partition algorithm of the existing system time complexity is $O(n^2)$, whereas the partition algorithm of the proposed system time complexity is O(n). Hence the proposed partitioning algorithm works out in less time compared to the existing partition algorithm. Second, the partition algorithm of the proposed system is effective in partitioning than the existing system which is depicted in the example discussed above.

The Table 1 shows the inter partition similarity value for both the systems. The proposed partition algorithm effectively partitions the ontology so that the number of partition pairs to be matched further is less. From the above example the number of similar partition pairs to be matched further for existing system is 5, whereas for proposed system is 4 and hence processing time for entity level matching is reduced more in proposed compared to the existing system.

Table 1. Inter Partition Similarity

Partition Pair	Existing Inter Partition Similarity	Proposed Inter Partition Similarity
(p1,p`1)	0.33	0.66
(p1,p`2)	0.5	0
(p2,p`1)	0.4	0.4
(p2,p`2)	0	0.4
(p3,p`1)	0.57	0
(p3,p`2)	0.4	0.4

6. Experimental Setup

The ontology dataset Mouse anatomy [18] and NCI anatomy [19] will be used for evaluating the proposed ontology matching algorithm. These datasets are chosen for 3 reasons. First, the number of entity in Mouse anatomy is 2744 and NCI anatomy is 3044 which is sufficient enough to be considered as large ontology. Second, the reference match result [20] is available for these anatomy dataset which is needed for the evaluation of the algorithm. Third the OAEI, the international Ontology Alignment Evaluation Initiative suggests these two anatomy ontologies for evaluating the large scale ontology matching algorithm. The JENA library [21] is used to preprocess the input anatomy ontologies.

The performance metrics used for evaluation are precision, recall and time taken by the ontology matching algorithm to output the match result. Precision is a measure of correctness and Recall is a measure of completeness which are defined as follows.

$$Precision=|A \cap R| / |A| \quad (11)$$

$$Recall=|A \cap R| / |R| \quad (12)$$

where A is the set of match results of the proposed ontology matching algorithm and R is the set of reference match results. The following parameter should be determined experimentally.

- Parameter CutOff which is used to choose partitions for merging among a set of partitions.
- Parameter β indicating the minimum cohesion of the partition set. If the cohesion of the partition set goes below β the partitioning algorithm terminates.
- Experimentally determine the best set of entity neighbours. The possible neighbours are ancestors, descendants, siblings, grand ancestors and grand descendants.

To prove the increased efficiency of the proposed structural proximity computation, the existing structural proximity computation of AHSCAN structural proximity computation is implemented. The neighbourhood of AHSCAN which is used in structural proximity computation can be defined as follows.

$$N(e)= \{w \in E \mid (e,w) \in R\} \text{ and } \{e\} \quad (13)$$

where e and w are entity of the entity set E and R is the relation set.

$$\sigma(ei,ej) = \frac{|N(ei) \cap N(ej)|}{\sqrt{|N(ei)|.|N(ej)|}} \quad (14)$$

where e_i and e_j are two entities of ontology O, $N(e_i)$ is defined by equation 13, $|N(ei) \cap N(ej)|$ represents the number of common entities between their contexts and sqrt($|N(ei)|.|N(ei)|$) is the geometric mean of the two contexts size used to normalize the value of the structure similarity.

The structural proximity measure of AHSCAN, LOMPT structural proximity and LOMPT structural proximity with alpha = 0.5 are almost similar because numerator is same for all three i.e. all consider the common feature between the entities. They differ in the denominator part where geometric mean of the number of features of the two entities is computed for structural proximity measure of AHSCAN, average mean is computed for LOMPT structural proximity with alpha = 0.5 and weighted mean based on alpha is computed for LOMPT structural proximity .

The experiment on these three structural proximity measures depicts that, all these three measure correlate well with each other meaning that all these three measure are equally effective. Hence the LOMPT structural proximity with alpha = 0.5 is used for further experiments since its computation cost is theoretically less compare to other two measure. The sample ontology used for the experiment which is taken from [22] is shown in Fig. 9. The Fig. 10 and 11 depicts the entity wise correlation between the intra ontology similarity measures with the help of the graph.

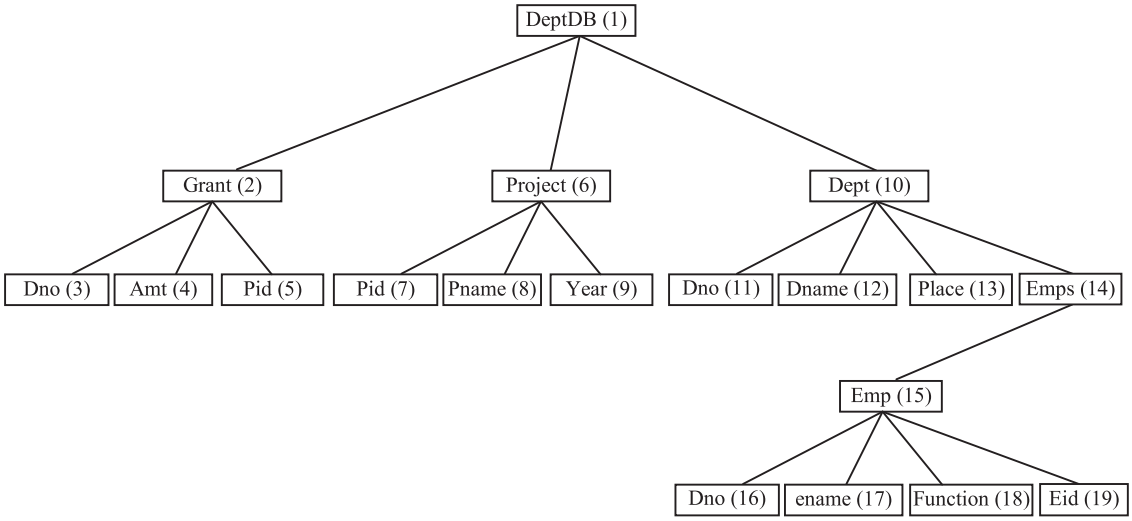

Fig. 9. Sample ontology for experiment

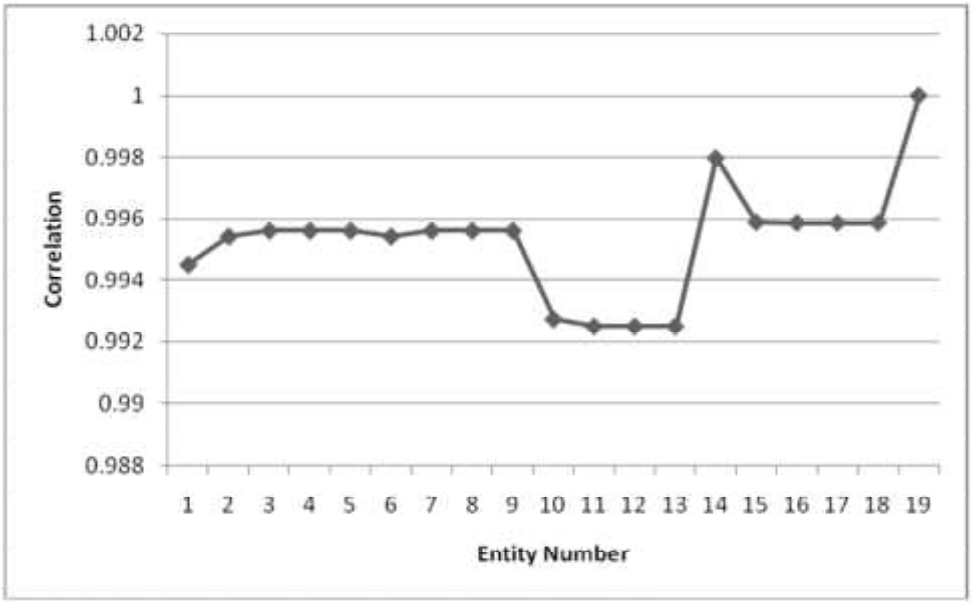

Fig. 10. Correlation between LOMPT structural proximity with alpha = 0.5 and LOMPT structural proximity

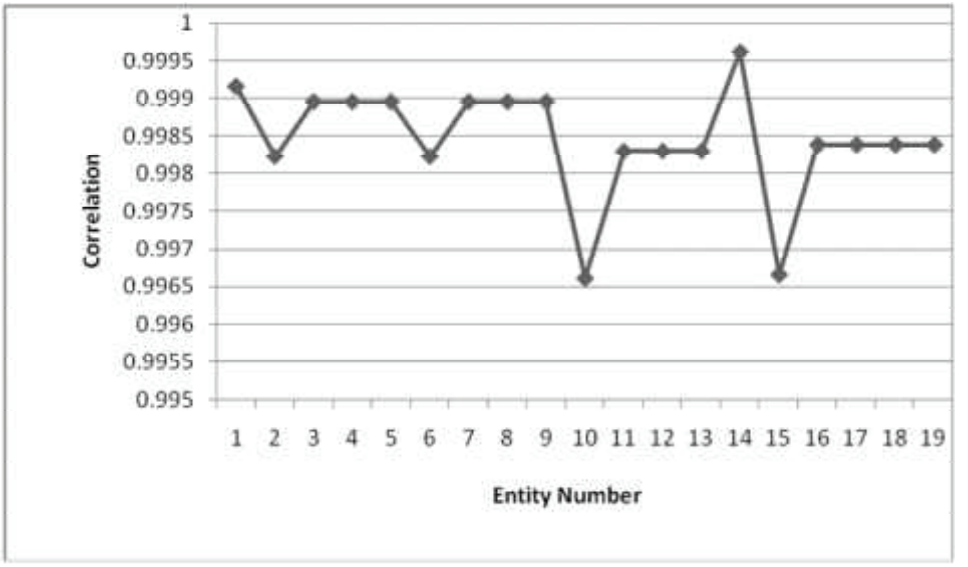

Fig. 11. Correlation between LOMPT structural proximity with alpha = 0.5 and AHSCAN structural proximity

7. Conclusion and Future Enhancements

This paper has proposed a partition based ontology matching algorithm which can efficiently handle large ontologies. The algorithm is designed in such a way that the efficiency is improved without compromising on effectiveness. In particular a new neighbour based structural proximity measure is devised which will decrease the time taken since the structural proximity are calculated only between the neighbours. Further the efficiency of the LOMPT ontology matcher is appreciable as the proposed partitioning algorithm is linearly scalable. Moreover the proposed Si-SUB light weight string matcher increases efficiency as the number of comparisons is reduced.

In future, experiments can be conducted to reduce the time taken for discovering anchors by using even more light weight string matcher than the proposed one or by random picking of the entities. Similarly by experiment, the time taken for finding similar partition pairs can also be reduced by constructing virtual documents for each partition.

References

[1] Shvaiko P and Euzenat J. Ten challenges for ontology matching. In: *Proceedings of the Move to Meaningful Internet Systems*, 2008, p. 1164–1182.

[2] O. A. E. Initiative, (2010). http://20.ontologymatching.org/.

[3] Rahm E. Towards Large-Scale Schema and Ontology Matching. In: Bellahsene Z, Bonifati A, Rahm E, editors. *Schema matching and mapping*, New York: Springer Heidelberg, 2011, p. 3–27.

[4] Hanif MS, Aono M. An efficient and scalable algorithm for segmented alignment of ontologies of arbitrary size. *Journal of Web Semantics*, 2009, Vol. 7, No. 4, p. 344–356.

[5] Do HH, Rahm E. Matching large schemas: Approaches and evaluation. *Journal of Information System*, 2007, Vol. 3, No. 6, p. 857–885.

[6] Hu W, Qu Y, Cheng G. Matching large ontologies: A divide-and-conquer-approach. *Journal of Data Knowledge Engineering*, 2008, Vol. 67, No. 1, p. 140–160.

[7] Hamdi F, Safar B, Reynaud C, Zargayouna H. Alignment-based partitioning of large-scale ontologies. *Advances in knowledge discovery and management*, 1st ed. New York: Springer Heidelberg, 2009, p. 251–269.

[8] P.F. Patel-Schneider, P. Hayes, I. Horrocks (Eds.). OWL web ontology language semantics and abstract syntax. W3C Recommendation, 10 February 2004. <http://www.w3.org/TR/owl-semantics/>.

[9] Ehrig M, Staab S. Quick ontology matching. In: *Proceedings of international conference semantic web (ICSW)*. LNCS, vol 3298. Springer, Heidelberg, 2004, p. 683–697.

[10] Peukert E, Berthold H, Rahm E. Rewrite techniques for performance optimization of schema matching processes. In: *Proceedings of 13th international conference on extending database technology (EDBT).* ACM, NY, 2010, p. 453–464.

[11] Gross A, Hartung M, Kirsten T, Rahm E. On matching large life science ontologies in parallel. In: *Proceedings of 7th international conference on data integration in the life sciences (DILS)*. LNCS, 2010, Vol. 6254. Springer, Heidelberg.

[12] Li J, Tang J, Li Y, Luo Q. RiMOM:A dynamic multistrategy ontology alignment framework. In: *IEEE Trans Knowl Data Eng* , 2009, Vol. 21, No. 8, p. 1218–1232.

[13] Guha S, Rastogi R, Shim K. ROCK: a robust clustering algorithm for categorical attributes. In: *Proceedings of the 15th International Conference on Data Engineering*, 1999, p. 512–521.

[14] A. Tversky. Features of similarity. *Psychological Review*, 1977, Vol. 84, p. 327-352.

[15] N. Yuruk, M. Mete, X. Xu, and T. A. J. Schweiger. AHSCAN: Agglomerative hierarchical structural clustering algorithm for networks. In: *International Conference on Advances in Social Network Analysis and Mining*, 2009, p. 72-77.

[16] Stoilos G, Stamou G, Kollias S. A string metric for ontology alignment. In: *Proceedings of the 4th International Semantic Web Conference*, LNCS, Vol. 3729, Springer, 2005, p. 624–637.

[17] W. Winkler. The state record linkage and current research problems. *Technical Report, Statistics of Income Division, Internal Revenue Service Publication, 1999*.

[18] Mouse_Anatomy (2011), Available at: seals-test.sti2.at/tdrs- web/testdata/persistent/ca1cddfe-c728-4207-b33d-ca245642f4c9/712477b7-e1f9-4633-8d52-17d2b25af008/suite/anatomy-track1/component/source

[19] NCI_Anatomy (2011), Available at : hseals-test.sti2.at/tdrs-web/testdata/persistent/ca1cddfe-c728-4207-b33d-ca245642f4c9/712477b7-e1f9-4633-8d52-17d2b25af008/suite/anatomy-track1/component/target

[20] Reference Alignment (2011), Available at: seals-test.sti2.at/tdrs-web/testdata/persistent/ca1cddfe-c728-4207-b33d-ca245642f4c9/712477b7-e1f9-4633-8d52-17d2b25af008/suite/anatomy-track1/component/reference

[21] JENA API, Available at: http://jena.sourceforge.net/.

[22] L. Chiticariu, M. A. Hernndez, P. G. Kolaitis, L. Popa. Semi-automatic schema integration in Clio. In: *VLDB'07*, 2007, p. 1326–1329.